Eyal Orbach - id 015369317
Daniel Juravski - id 206082323

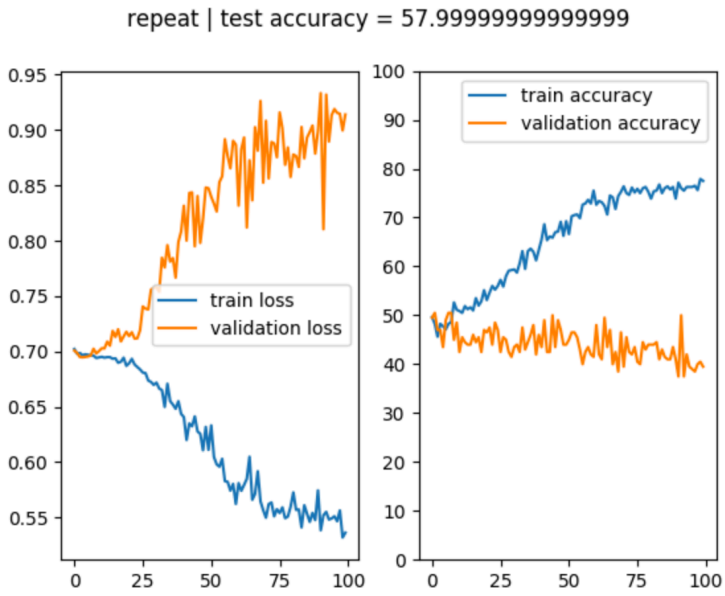
1. Repeating sequence

Language description:
Vocabulary = {1, 2, 3, 4, 5, 6, 7, 8, 9,a , b, c, d}
Valid sequence (in the language): A sequence made from the above characters which is a concatenation of 2 identical sequences
example = “a13a13”
Invalid sequence (outside the language): Anything else
example = “a13a14”

Why should it fail:
To correctly resolve if a sequence repeats itself while iterating only once on the characters the module must memorize the characters it has seen, since there is a limit on that memory size we could always generate a sequence too large for it to remember enough characters.

Experiment results:
1000 examples.
100 epochs on all examples.
LSTM fails with no signs of improvement

Initial examples were completely random sequences for the invalid sequences, which resulted in the LSTM finding some rules that fitted the train data, enabling the train accuracy to increment but still reaching ~50% on validation and test data.
We have then modified the invalid examples to be similar to the valid examples with one random letter modified, thus making it a non repeating sequence.
This caused the LSTM to stay stuck around 50% at train time also.



2. Char at Index

Language description:
Vocabulary = {1, a , b, c, d}
Valid sequence (in the language): A sequence which is a concatenation of 2 parts
part 1 = random sequence from {a , b, c, d}
part 2 = a sequence of 1's, followed by a single character from {a , b, c, d}
where the number of 1's is an index where that letter can be found in the first part:
example = "abcd111c"
(1 appears 3 times and c can be fount at index 3)

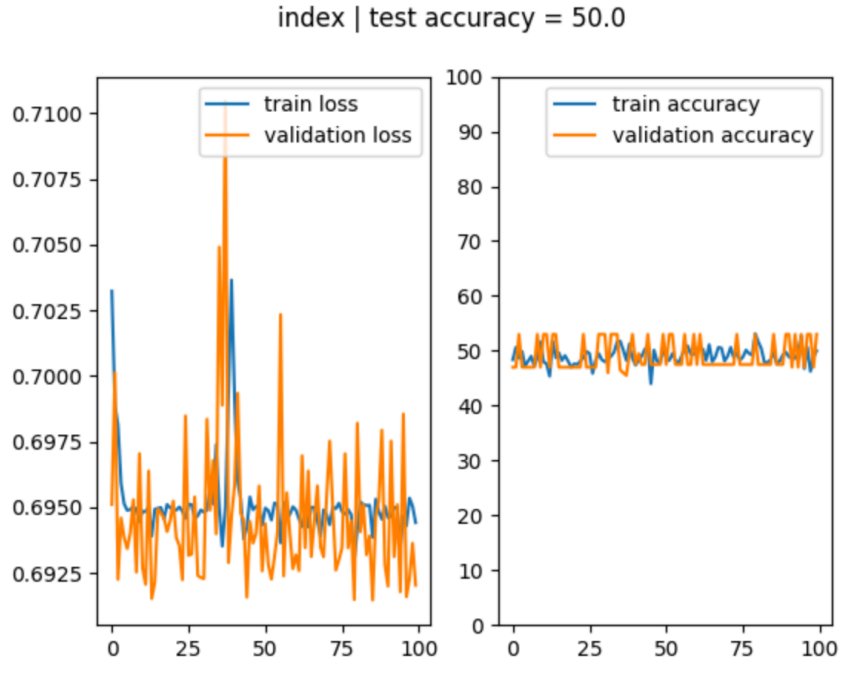
Invalid sequence (outside the language): Anything else
example = "abcd111a"

Why should it fail:
To correctly resolve this the model needs to not only understand this rule but remember the whole sequence before reaching the last letter so it can check if that is the letter in that index. Because of the memory limitations discussed earlier this is not possible.

Experiment results:
1000 examples.
100 epochs on all examples.
LSTM fails with no signs of improvement

We have also generated a slightly different expirement which we thought would be possible for it to learn. Having the letter and sequence of 1's be before the random letters sequence:
example "c111abcd".
This needs only constant memory to be solved (representing first letter and some counter) but the LSTM did not succeed in learning that either.

The graph describes the experiment with the initial index problem, the one we described as unsolvable (the easier problem with the 1's sequence at the beginning produced similar graph though)



2. Is Prime Number

Language description:
Vocabulary = {0,1,2,3,4,5,6,7,8,9}
Valid sequence (in the language): A sequence which is a decimal representation of a prime number.
example “149”

invalid sequence (outside the language): A sequence which is a decimal representation of a non-prime number
example = “49”
we have filtered numbers that divide by 2 or 5.

Why should it fail:
There is no known pattern to prime numbers other than their actual definition.
The concepts of division and natural numbers are not trivial for the module to represent and even if that was solved, there are no known states that the module can remember that will enable it to solve the problem character by character without remembering the complete number.

Experiment results:
1000 examples.
100 epochs on all examples.
LSTM learns the training data without real generalization.

Initially the LSTM returned rather nice-looking results, but we realized that it might rely on the last character in the sequence solving easily ~60 percent of examples (ending by 0,2,4,6,8,5). Once we filtered the examples to not include numbers divided by 2 or 5, The accuracy dropped significantly.